Notas de aula 2

Algoritmos de Ordenação - Introdução e Algoritmos Básicos

Prof. Élder F. F. Bernardi (elder.bernardi@passofundo.ifsul.edu.br)
Estrutura de dados III - Ciência da Computação
IFSul - Câmpus Passo Fundo

1. O problema da ordenação ou classificação em ordem

Ordenar ou classificar um conjunto de valores é um problema bastante comum e recorrente que implica em organizar um conjunto de valores ou elementos em uma nova **ordem** determinada, de acordo com uma **regra de comparação relativa**. Por exemplo: classificar um conjunto de palavras em ordem alfabética decrescente. Decrescente diz respeito à **ordem**. Alfabética é a **regra** de comparação. Uma regra de comparação deve estabelecer critérios onde possamos comparar um elemento do conjunto e classificá-lo em relação a outro similar como: **maior, menor** ou **igual**. Desta forma podemos estabelecer um ordenamento de um conjunto de valores, desde que estes sejam comparáveis entre si.

Segundo [Cormen, 2009], usando um exemplo de ordenação crescente de números, o problema de ordenação pode ser expresso da seguinte forma:

Entrada: uma sequência de *n* números $(a_1, a_2, a_3 \ldots a_n)$.

Saída: Um reordenamento da sequência de entrada $(a'_1, a'_2, a'_3, \ldots, a'_n)$, em que $a'_1 \leq a'_2 \leq a'_3 \leq \ldots \leq a'_n$.

Tal sequência é representada frequentemente como um vetor de tamanho n, embora possa ser representada de outras maneiras, tal como uma lista.

Embora seja um problema trivial, quando se trata de uma entrada muito grande, pode-se ter problemas que levam à escolha de soluções mais complexas. Para compreender melhor isto, faz-se necessária a apresentação dos algoritmos mais simples de ordenação, que são os temas desta aula.

Importante ressaltar que estudar os algoritmos de ordenação é, além de dominar a solução do problema de ordenação, compreender e exercitar algumas técnicas fundamentais dentro da computação, tais como: **divisão e conquista, subdivisão binárias, recursão e hashing.** E algumas estruturas de dados úteis, como **listas, heaps e árvores.**

Site para visualização dos algoritmos:
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Ordenação de bolha ou Bubble Sort

Este algoritmo de ordenação talvez seja válido somente como um mau exemplo. Por ter uma característica de loops de trocas, onde em cada interação não necessariamente se posiciona algum item a ser ordenado em uma posição definitiva, faz com que este algoritmo tenha um desempenho muito ruim para a imensa maioria dos casos práticos.

Seu algoritmo se baseia em realizar n laços partindo do primeiro até o último elemento do conjunto de ordenação e ir realizando trocas com os itens adjacentes a cada iteração, caso o item atual seja maior que o próximo (em caso crescente). Este processo é contínuo até que se encontre uma iteração cuja a qual não haja troca.

Poderia ser descrito desta maneira:

```
faça
      trocas = false;
      para cada item i, de i=0 até i <= n-2 //n
            se a'_i > a'_i+i
                  troque( a'_i, a_i+1)
                  trocas = true;
enquanto trocas == true //n
```

Sua complexidade é O(n^2), pois realiza dois laços aninhados onde no pior caso são realizadas n * n trocas. Este algoritmo somente pode ser utilizado em conjunto de dados muito pequenos, sendo que há alternativas mais eficientes para casos práticos.

Insertion Sort

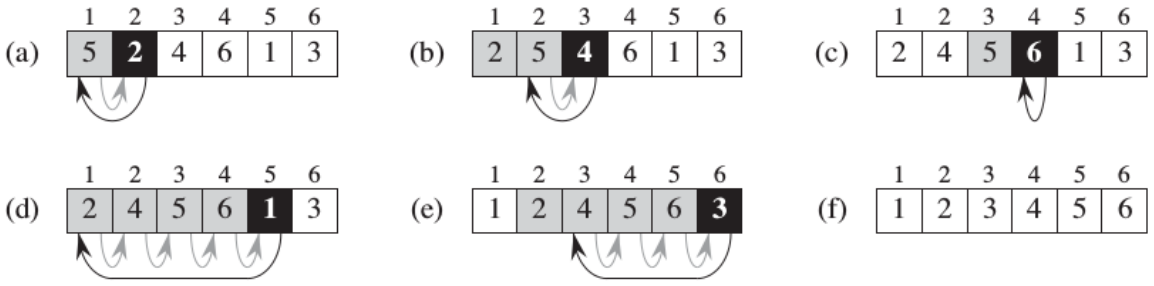
Este algoritmo tem uma premissa bastante simples:

```
para cada item até n:
      Encontre a posição onde o item pertence.
      Crie o espaço para a inserção, deslocando os demais itens
      Insira-o na posição.
```

Então por exemplo, dado o seguinte conjunto de elementos, os passos que acontecem são os seguintes:

Figura 1: passo da solução

Site para visualização dos algoritmos:
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html



Fonte: Tormen, 2009.

O mecanismo do algoritmo funciona com a escolha de um item, partindo de i = 1, sendo 1 <=i >n, onde se compara a_i com seus antecedentes até que se encontre qual a posição de a_i ordenada dentro do conjunto $a_{<=i}$, ou seja, até que não haja nenhum item menor do que o que está sendo comparado até o início do vetor. Então, posiciona-se a_i na posição encontrada. Para se ter este espaço, o algoritmo vai deslocando os elementos maiores em uma posição à direita, até que se encontre a posição adequada para a_i. Analogicamente, num jogo de cartas, pode-se comparar a ordenação de cartas que se faz à medida que se retira as cartas do baralho e se ordena na mão.

O algoritmo em alto nível do Insertion Sort, segundo [Cormen, 2009] pode ser descrito desta forma:

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 .. j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

Complexidade do algoritmo

O algoritimo é de complexidade O(n^2), pois para cada número a ser ordenado, faz-se a comparação com até n-1 elementos, portanto, tem-se n*n, caracterizando a classe do algoritmo.

Site para visualização dos algoritmos:
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

Selection Sort

O algoritmo Selection Sort é similar ao Insertion Sort, com a diferença de que ao invés de posicionar cada item *i* na sua posição de ordenação a partir de *n-i,* o selection faz uma busca, para cada i, do menor item da lista. Então o item é posicionado na posição i. Seria basicamente uma busca pelo menor item no conjunto de valores e seu posicionamento definitivo, feito até que não haja mais itens a serem buscados.

Seu algoritmo poderia ser expresso desta forma:

```
para i=1 em n
      encontre o menor item entre os não ordenados, dito k
      posicione o item em i, fazendo a troca do item k com i
```

Tal algoritmo tem a mesma complexidade do Insertion Sort, porém, em casos práticos seu desempenho é menor, pois enquanto o insertion sort somente percorre um conjunto limitado de valores até posicionar um item, o selection sort irá sempre percorrer toda a lista fazendo comparações para posicionar cada elemento.

Atividade:

1 - Implemente os algoritmos bubble, selection e insertion sort. Crie uma forma de gerar vetores aleatórios e aplique a ordenação destes vetores com cada algoritmo. Conte a quantidade de trocas realizadas por cada um, em cada caso para fins de termos uma comparação empírica de cada caso.

2 - Utilize uma estrutura de dados do tipo lista encadeada ao invés de um vetor e compare a implementação do insertion sort em um vetor e na lista.

Site para visualização dos algoritmos:
https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html